

In diesem Projekt entwickelst du schrittweise und über einen längeren Zeitraum ein Programm zur Verwaltung des **Wartezimmers einer Arztpraxis**.

Du nimmst die Rollen „Programmierer\*in“ und „Tester\*in“ ein. Das heißt, um Anforderungsanalyse und Design brauchst du dich nicht zu kümmern, die sind bereits fertig und bilden die Grundlage der Aufgabe.



## Anforderungen

Als Programmierer\*in musst die Anwendungssituation verstehen und die Anforderungen an das Programm kennen, also die Funktionen, die von den Benutzern des Programms gewünscht sind.

### Benutzer

Das Programm wird von Arzthelfer\*innen bedient, am Empfang einer Arztpraxis. Patient\*innen und Ärzt\*innen sind als Benutzer nicht vorgesehen.

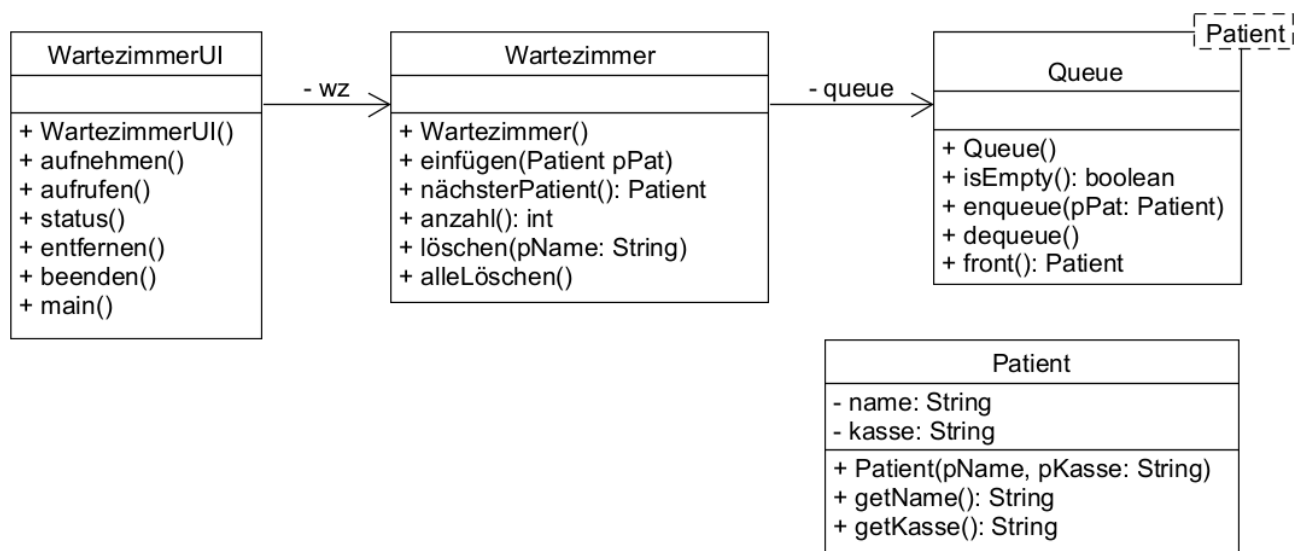
**Anwendungsfälle** (Situationen, in denen das Programm benutzt wird)

- **Aufnehmen:** Patient\*innen, die die Praxis betreten, werden von Arzthelfer\*innen mündlich nach ihrem Namen und ihrer Krankenkasse gefragt. Dann werden diese Daten in das Programm eingegeben, und die Patient\*innen setzen sich ins Wartezimmer.
- **Aufrufen:** Wenn die Arzthelfer\*innen jemand für die nächste Behandlung aufrufen sollen, schauen sie im Programm, wer als nächstes an der Reihe ist. Das Programm gibt den entsprechenden Namen und die Krankenkasse der Patientin bzw. des Patienten aus.
- **Status:** Das Programm soll ausgeben, wie viele Patient\*innen zur Zeit warten.
- **Patient\*in entfernen:** Falls jemand die Behandlung nicht abwarten kann und vorher nach Hause gehen möchte, kann das Programm diese Person aus der Liste der Wartenden entfernen.
- **Beenden:** Falls die Praxis schließt, bevor alle wartenden Patient\*innen behandelt wurden, werden die verbleibenden aus der Liste gelöscht.

## Design

Beim Entwurf der Software soll darauf geachtet werden, dass **Funktionen** und **Benutzerschnittstelle** voneinander getrennt bleiben. So wird sichergestellt, dass man die Benutzerschnittstelle gegebenenfalls austauschen kann, ohne die Funktionen neu programmieren zu müssen.

Aus diesem Grundsatz ergibt sich das folgende Klassendiagramm:



In der Hauptklasse **WartezimmerUI** („user interface“ = Benutzerschnittstelle) wird die gesamte Interaktion mit den Benutzern implementiert.

Wie in den bisherigen Aufgaben verwenden wir dazu Eingaben und Ausgaben mit der Konsole.

In der Klasse **Wartezimmer** werden alle Funktionen – einfügen, entfernen usw. – implementiert. Die Methoden dieser Klasse enthalten keine Ein- und Ausgaben mit der Konsole.

Objekte der Klasse **Patient** enthalten für das Programm relevante Daten der einzelnen Patienten. Sie sind für dieses Projekt stark reduziert. Der Fokus liegt hier nicht darauf, eine realistische Patientenverwaltung zu entwickeln, sondern mit einer Datenstruktur (hier der Queue) zu arbeiten.

Die **Queue** wird als Datenstruktur zur Aufbewahrung der Patienten-Objekte verwendet, da das Wartezimmer nach dem „first in, first out“ Prinzip funktioniert.

Eine Dokumentation der einzelnen Methoden dieser Klassen findest du im Anhang.

Die Klasse Queue kennst du bereits, verwende ggf. in das Material aus dem vorigen Abschnitt.

## Arbeitsschritte

In größeren Projekten ist es nicht sinnvoll, alle Klassen vollständig zu implementieren und das Programm erst zum Schluss zu kompilieren und zu testen. Stattdessen teilt man die Arbeit in mehrere Schritte ein. In jedem Schritt implementierst du einen Teil der Funktionen des Programms. Das Programm wird am Ende jedes Schrittes kompiliert, ausgeführt und getestet. Erst, wenn dieser Teil des Programms funktioniert, beginnst du mit dem nächsten Schritt.

## Schritt 1: Menü

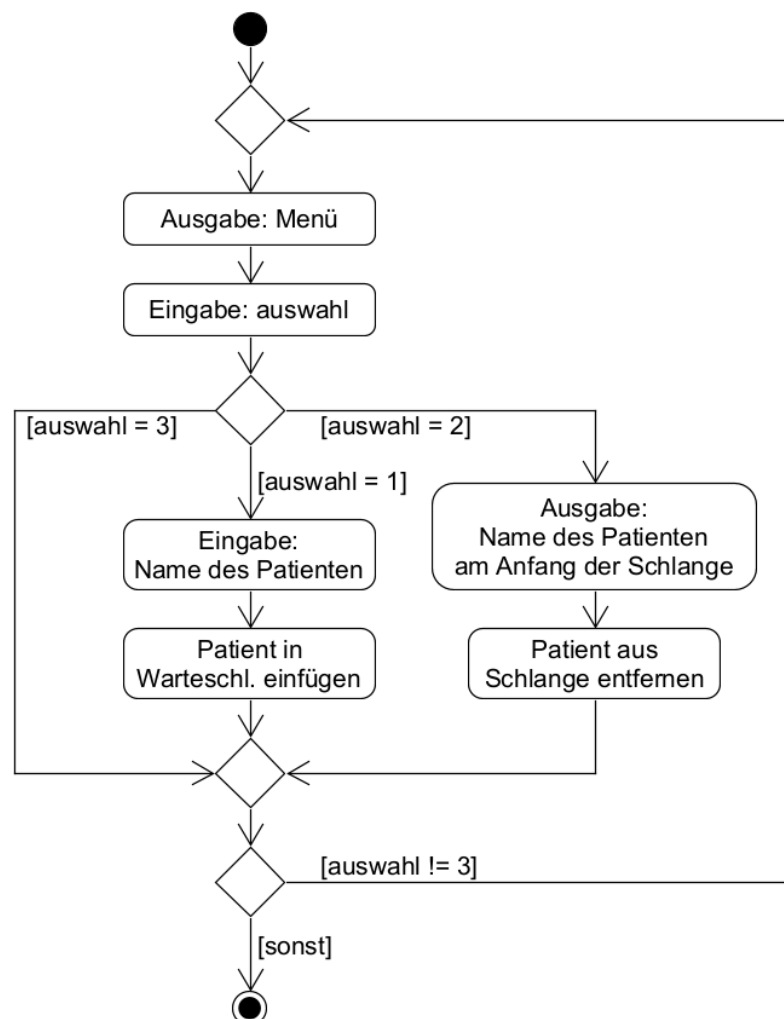
Implementiere als erstes die **main-Methode**, die den Ablauf des Programms steuert. Du arbeitest also zuerst an der Benutzerschnittstelle, das heißt hier: mit der Konsole.

Bsp. für das **Menü**:

(die Funktionen  
werden später noch  
erweitert)

```
--- Menü ---  
1. Neuen Patient aufnehmen  
2. Nächsten Patient zur Behandlung aufrufen  
3. Ende  
Bitte geben Sie ihre Wahl ein:
```

Der **Ablauf**  
des Programms:



## Tipp: Die do-while-Schleife

Die **while-Schleife** überprüft jeweils ZU BEGINN jeder Wiederholung, ob die Bedingung erfüllt ist. Falls dem so ist, geht es in die (erste, bzw. die nächste) Wiederholung.

Die while-Schleife wird auch „kopfgesteuerte Schleife“ genannt.

Manchmal ist das jedoch ungünstig: das Menü soll **mindestens einmal** angezeigt werden und der Benutzer soll eine Auswahl treffen. Wenn er dann die Auswahl trifft, das Programm zu beenden, bricht die Schleife ab, sonst wird das Menü wieder angezeigt.

Für solche Fälle kann man die **do-while-Schleife** verwenden. Dabei wird die Bedingung AM ENDE jeder Wiederholung geprüft, so dass sie mindestens einmal ausgeführt wird.

Die do-while-Schleife wird auch „fußgesteuerte Schleife“ genannt.

### Beispiel mit while-Schleife:

```
public void main()
{
    int wahl;
    wahl = 0;           // sicherstellen, dass Bedingung erfüllt ist
    while (wahl != 3)
    {
        Console.println("Menü");
        Console.println("1: Neuen Patient aufnehmen");
        Console.println("2: Patient aufrufen");
        Console.println("3: Beenden");

        wahl = Console.readInt();
        if (wahl == 1) { aufnehmen(); }
        else if (wahl == 2) { aufrufen(); }
    }
}
```

### Beispiel mit do-while-Schleife:

```
public void main()
{
    int wahl;
    do
    {
        Console.println("Menü");
        Console.println("1: Neuen Patient aufnehmen");
        Console.println("2: Patient aufrufen");
        Console.println("3: Beenden");

        wahl = Console.readInt();
        if (wahl == 1) { aufnehmen(); }
        else if (wahl == 2) { aufrufen(); }

    } while (wahl != 3);
}
```

## Schritt 2: Grundfunktionen

Implementiere die Funktionen „neuen Patient aufnehmen“ und „Patient aufrufen“.

Die Funktionen, die du in der Klasse **Wartezimmer** implementieren musst, sind dabei sehr einfach:

- Die Methode `void einfügen()` erhält ein bereits fertiges Patienten-Objekt als Parameter und hängt dieses einfach ans Ende der Queue.
- Die Methode `Patient nächsterPatient()` gibt das Patienten-Objekt am Anfang der Queue zurück und löscht es aus der Queue. Falls die Queue leer ist, wird null zurückgegeben.

Die eigentliche Arbeit liegt in der Benutzerschnittstelle, in der Klasse **WartezimmerUI**:

- Die Methode `void aufnehmen()` fordert den Benutzer über die Konsole auf, Namen und Krankenkasse des neu eingetretenen Patienten einzugeben. Damit wird ein Objekt der Klasse `Patient` erzeugt und der Methode `einfügen()` als Parameter übergeben.
- Die Methode `void aufrufen()` holt sich mit der Methode `nächsterPatient()` das Objekt des Patienten, der jetzt an der Reihe ist, und gibt dessen Namen aus. Es muss auch der Sonderfall beachtet werden, dass das Wartezimmer leer ist.

## Schritt 3: Erweiterung „Alle Löschen“

Das Programm wird um die eine Funktion erweitert: Die Warteschlange vollständig zu leeren, wenn das Programm beendet wird (z.B. am Ende eines Arbeitstages).

Die Funktion stellt die Methode `alleLöschen()` der Klasse `Wartezimmer` zur Verfügung.

Der entsprechende Teil der Benutzerschnittstelle wird in der Methode `beenden()` implementiert.

- a) Implementiere die Methode `alleLöschen()` der Klasse `Wartezimmer`.
- b) Implementiere die Methode `beenden()` der Klasse `WartezimmerUI`.  
Sie ruft die Methode `alleLöschen()` auf und gibt eine Botschaft auf der Konsole aus, z.B. „Das Programm wird beendet. Alle Patienten wurden aus der Warteschlange gelöscht.“
- c) In der `main`-Methode soll die Methode `beenden()` beim entsprechenden Menüpunkt aufgerufen werden.

## Schritt 4: Erweiterung „Status“

Um abschätzen zu können, wie viel Arbeit noch zu erledigen ist, sollte das Programm ausgeben können, wie viele Patient\*innen derzeit warten (den „Status“ des Wartezimmers).

Die Klasse **Wartezimmer** erhält dazu eine neue Methode `int anzahl()`, die zurückgibt, wie viele Objekte in der Queue gespeichert sind. Da die Klasse Queue selbst nicht „mitzählt“, wie viele Objekte eingefügt bzw. entfernt wurden, musst du diese Funktion in der Klasse Wartezimmer implementieren.

Die Klasse **WartezimmerUI** erhält ebenfalls eine neue Methode `status()`, die die Anzahl der wartenden Patient\*innen auf der Konsole ausgibt.

- a) Da diese und die weiteren Funktionen komplexer sind, entwickle für die Methode `anzahl()` zunächst einen **Algorithmus**. Stelle diesen in Form eines **Aktivitätsdiagramms** dar. (Hilfestellung: siehe Anhang)

Die Schwierigkeit liegt darin, dass du keinen direkten Zugriff auf Objekte der Queue (außer dem ersten) hast. Es müssen aber alle Objekte der Queue gezählt werden. Dein Algorithmus muss dazu Elemente vom Anfang der Queue entfernen. Am Ende des Algorithmus soll die Queue aber genau die gleiche Struktur haben wie zu Beginn des Algorithmus.

Achte auch auf mögliche Sonderfälle (z.B. „Schlange ist leer“).

- b) Implementiere die Methode `anzahl()` der Klasse Wartezimmer.
- c) Implementiere die entsprechende Methode `status()` der Klasse WartezimmerUI.
- d) Erweitere die main-Methode um einen neuen Menüpunkt „Status anzeigen“.

## Schritt 5: Erweiterung „Entfernen“

Wenn ein Patient die Praxis verlässt, bevor er behandelt wurde, soll das Programm eine Möglichkeit bieten, diesen Patienten aus der Warteschlange zu entfernen. Das Programm fordert den Benutzer (also den Arzthelfer bzw. die Arzthelferin) auf, den Namen einzutippen und entfernt ihn dann.

Zur Vereinfachung nehmen wir an, dass keine doppelten Namen vorkommen.

Die Methode `boolean löschen(String pName)` der Klasse **Wartezimmer** soll die gewünschte Funktion zur Verfügung stellen. Sie erhält den Namen des zu löschenden Patienten als Parameter, durchsucht die Queue nach einem Patienten-Objekt mit diesem Namen und entfernt dieses. Dabei soll die Struktur der Queue ansonsten erhalten bleiben.

In dem Sonderfall, dass es kein Objekt mit diesem Namen in der Queue gibt, gibt die Methode `false` zurück, ansonsten bei erfolgreicher Löschung `true`.

Der entsprechende Teil der Benutzerschnittstelle wird in der Methode `entfernen()` der Klasse **WartezimmerUI** implementiert. Dabei wird der Benutzer aufgefordert, den Namen des zu löschenden Patienten einzugeben. Anschließend wird die Methode `löschen()` aufgerufen, und es wird eine Bestätigung oder eine Fehlermeldung ausgegeben.

- a) Entwickle zuerst einen **Algorithmus** für die Methode `löschen()`.
- b) **Implementiere** dann die Methode `löschen()` der Klasse **Wartezimmer**.

Zur Erinnerung:

Anders als z.B. `int`- oder `double`-Werte können **Strings nicht mit `==` oder `!=` verglichen werden**. Bei Strings handelt es sich um Objekte, für die der Inhalt (also die Buchstaben) verschiedener Objekte verglichen wird. Dafür hat die Klasse `String` die Methode **`equals()`**. Diese gibt `true` zurück, wenn zwei Strings übereinstimmen.

- c) Implementiere die entsprechende Methode `entfernen()` der Klasse **WartezimmerUI**.
- d) Erweitere die `main`-Methode um einen neuen Menüpunkt „Patient entfernen“.

## Glückwunsch!

Du hast das Projekt erfolgreich abgeschlossen.

## Anhang: Dokumentation

### Klasse WartezimmerUI („user interface“ = Benutzerschnittstelle)

In dieser Klasse wird die Interaktion mit den Benutzern – hier mit der Konsole – implementiert.

- Der Konstruktor erzeugt das Wartezimmer-Objekt.
- `void aufnehmen()`  
Die Konsole bittet um Eingabe des Names und der Krankenkasse eines neu eingetretenen Patienten. Es wird ein Objekt der Klasse Patient mit diesen Daten erzeugt. Dieses wird der Methode `einfügen()` der Klasse Wartezimmer als Parameter übergeben.
- `void aufrufen()`  
Über die Methode `nächsterPatient()` der Klasse Wartezimmer wird abgefragt, welcher Patient als nächstes an der Reihe ist (dabei wird der Patient auch aus dem Wartezimmer gelöscht). Die Daten dieses Patienten werden auf der Konsole ausgegeben.
- `void status()`  
Mithilfe der Methode `anzahl()` der Klasse Wartezimmer wird die Anzahl der wartenden Patienten auf der Konsole ausgegeben.
- `void entfernen()`  
Die Konsole bittet um Eingabe des Namens des zu löschenden Patienten und löscht ihn mithilfe der Methode `löschen()` der Klasse Wartezimmer. Falls der Patient nicht in der Warteschlange steht, wird eine Fehlermeldung ausgegeben.
- `void beenden()`  
Alle Patienten werden aus der Warteschlange entfernt.
- `void main()`  
Die `main`-Methode gibt in einer Schleife ein Konsolen-Menü aus, das alle Funktionen des Programms auflistet, und bittet den Benutzer, eine dieser Funktionen auszuwählen. Nach der Auswahl wird die entsprechende Funktion ausgeführt. Danach wird das Konsolen-Menü wieder angezeigt. Das ganze wiederholt sich solange, bis der Benutzer die Funktion „Beenden“ wählt.

### Klasse Wartezimmer

In dieser Klasse werden die Funktionen des Programms – einfügen von Patienten, entfernen usw. – implementiert. Entsprechend der Trennung von Benutzerschnittstelle und Funktionen werden in dieser Klasse **keine Methoden der Klasse Console** verwendet.

- Der Konstruktor erzeugt das Queue-Objekt.
- `void einfügen(Patient pPat)`  
Das im Parameter übergebene Patienten-Objekt wird ans Ende der Queue gehängt.
- `Patient nächsterPatient()`  
Falls es Patienten in der Warteschlange gibt, wird das Objekt am Anfang der Queue gelöscht und als Rückgabewert zurückgegeben. Andernfalls gibt die Methode `null` zurück.
- `int anzahl()`  
Die Methode zählt, wie viele Patienten die Queue enthält, und gibt diese Zahl zurück. Die Queue hat am Ende der Methode die gleiche Struktur wie vorher.



- `boolean löschen(String pName)`  
Die Queue wird nach einem Patienten-Objekt durchsucht, dessen Name mit `pName` übereinstimmt. Dieses wird aus der Queue entfernt. Die restlichen Patienten bleiben in der gleichen Reihenfolge wie vorher. Falls das Objekt gefunden und gelöscht wurde, wird `true` zurückgegeben, sonst `false`.
- `void alleLöschen()`  
Die Queue wird vollständig geleert, die enthaltenen Objekte dabei gelöscht.

### Klasse Patient

Objekte dieser Klasse enthalten die für das Programm relevanten Daten der einzelnen Patienten.

- Beim Erzeugen eines Patienten-Objekts werden dem Konstruktors der Name des Patienten und seine Krankenkasse mit den Parameter `pName` und `pKasse` übergeben.
- `String getName()`  
Gibt den Namen des Patienten zurück.
- `String getKasse()`  
Gibt die Krankenkasse des Patienten zurück.

**Anhang: Beispiel Aktivitätsdiagramm**